Towards the Automatic Generation of Card Games through Grammar-Guided Genetic Programming

Jose M. Font
Universidad Politecnica de Madrid, Spain
jm.font@upm.es

Tobias Mahlmann
IT University of Copenhagen, Denmark
tmah@itu.dk

Daniel Manrique
Universidad Politecnica de Madrid, Spain
dmanrique@fi.upm.es

Julian Togelius
IT University of Copenhagen, Denmark
julian@togelius.com

ABSTRACT

We demonstrate generating complete and playable card games using evolutionary algorithms. Card games are represented in a previously devised card game description language, a context-free grammar. The syntax of this language allows us to use grammar-guided genetic programming. Candidate card games are evaluated through a cascading evaluation function, a multi-step process where games with undesired properties are progressively weeded out. Three representative examples of generated games are analysed. We observed that these games are reasonably balanced and have skill elements, they are not yet entertaining for human players. The particular shortcomings of the examples are discussed in regard to the generative process to be able to generate quality games.

1. INTRODUCTION

Can a computer program design a complete game? The two main game genres, where automatic game design has been attempted, are board games and simple 2D arcade games. In board games, the early work of Hom and Marks [9] was followed by Browne's *Ludi* system which managed to evolve a board game of sufficient novelty and quality to be sold commercially [1]. A system generating simple arcade games was proposed by Togelius and Schmidhuber [15] which was followed up by work by Smith and Mateas [13] and Cook and Colton [3]. In a related genre, Mahlmann et al. have defined a GDL for turn-based strategy games [11]. In all of these examples, the space of games was searched using evolutionary algorithms, except Smith and Mates who use answer set programming and constraint solving. An overview of research on modelling and generating game rules can be found in a recent tutorial by Nelson [12].

A central question when generating games is how to represent a potential solution. There are several important considerations when devising a rule representation system, or a Game Description Language (GDL), for a particular game genre: the expressivity of the language (all games that you wish to be able to generate should indeed be expressible in the language), the compactness of the language, its human-readability, and how easily it can be searched by a search algorithm such as an evolutionary algorithm. In this context, rules can be seen as a type of content.

A second question when generating games is how to automatically evaluate generated games; this is particularly important when using a search-based approach such as artificial evolution, but also games generated e.g. by a solver-based approach such as answer set programming need to be evaluated at some point. Unlike many other types of content, it seems very hard to do any meaningful evaluation at all without actually playing the game. This leaves us with the question of how to play a completely unseen game. This question has been investigated within the General Game Playing competition, where algorithms are tasked with playing a number of unseen human-designed games (specified in a very detailed and verbose GDL) [10]. While various techniques have been attempted, the only technique that had notable success is Monte-Carlo Tree Search (MCTS). MCTS is a recently developed game-playing algorithm with very broad applicability [2].

This paper builds on a recent paper, which defined the core of a GDL for card games, and showed that it could be used to describe and generate working versions of the well-known card games blackjack, UNO and a somewhat simplified version of Texas hold 'em poker [5]. The language is defined as a context-free grammar, where each derivation is a syntactically valid game description which can be used to generate a game which is "playable" (it does not break the game engine). We use the previously defined card GDL to generate complete card games using an evolutionary algorithm and a set of specially designed fitness functions partially based on simulated playouts using both random play and MCTS which we combine through cascading elitism. We are not yet attempting directly to create entertaining games. We have instead set ourselves a couple of more modest targets which we regard as almost-necessary but not sufficient conditions for entertaining games, namely games that: (1) Can

be won or lost within a reasonable time; (2) usually present a choice of legitimate moves to each player each turn; (3) do not usually end in a draw; (4) do not unfairly advantage any particular player (position around the table); and (4) reward skilled play over random play.

2. THE CARD GAME DESCRIPTION LANGUAGE

We developed a context-free grammar called $G_{cardgame}$ to generate a description language for card games [5]. This means that every word in the language codifies a set of features and specifications that define a card game. Besides their own rules, all card games contained in this language share a fixed set of basic axioms, that represent the main components of a game. These axioms define the the number of players in the game (P), the locations where card can be placed during the gameplay (L) and the locations where tokens can be places (K). Tokens are virtual representations of coins or chips that can be bet during the gameplay. Possible card locations are players' hands (H), a standard French deck of cards (D) and face-up table locations (T).

Every game in the language defines its own rules that, adhering these axioms, compose a playable card game. A card game is defined as a set of stages, a ranking and a set of winning conditions. Every stage comprises a set of conditional rules in the form "if antecedent then consequent". Antecedents are conditions that, when fulfilled, trigger actions specified in consequents which modify the game state. The ranking specifies the values of every card and card combination in the game. Winning conditions determine which player wins the game.

This grammar was intentionally designed with a set of high-level instructions, in order to allow the generation of a great variety of original games as well as known card games. The grammar $G_{cardgame}$ is capable of modelling Texas hold 'em poker, blackjack and UNO; it has been verified before, that many random expansions of the grammar are playable [5].

3. THE EVOLUTIONARY SYSTEM

Grammar-Guided Genetic Programming (GGGP) is an evolutionary technique that uses a context-free grammar to generate the language whose words are the whole set of individuals that codify a solution to a given problem [6, 4, 8, 7]. In this paper, GGGP on the grammar $G_{cardgame}$ is used to generate card games. Figure 1 provides an overview over the evolutionary system. First, a population of I individuals is created through a mixed initialization process. This means that the first three individuals in the population are the manually coded versions of Texas hold 'em poker, blackjack and UNO. The remaining individuals ($I-3$) are randomly generated following the restrictions defined by $G_{cardgame}$. This mixed initialization allows the evolutionary process to generate variations of both handcoded games and entirely novel ones. The population is evaluated using a cascading elitist fitness function inspired by [14]. This function performs four sequential evaluation steps for every individual, removing it from the population if it codifies a drawish, overfitted or non-playable game, and evaluating it otherwise. The algorithm implements a roullette wheel based operator, and both Whigham's crossover and mutation operators.

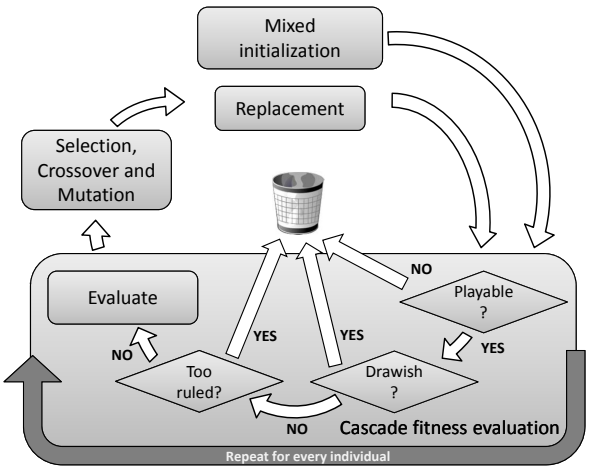

Figure 1: Description of the evolutionary system for generating card games.

The generated offspring is reinserted into the population. Notice that all individuals are grammatically valid, so there is no need to check if they are well formed. The evolutionary process ends when an individual (that is not Texas hold 'em poker, blackjack and UNO) with a target fitness score less or equal than a fixed score has been created.

The cascading evaluation function selects the pool of individuals to use for reproduction on the basis of a non-linear combination of four fitness evaluations [14]. E playouts are run on every individual's codified game. Players are simulated by *naive* agents that randomly choose their actions during the gameplay. The data gathered during these playouts is used to perform the following evaluations:

- If any game has not properly finished at least once, its codifying individual is removed from the population.
- Every individual whose game ended draw in more than the 20% of the playouts is also removed.
- If the game has a number of stages higher than 10, it is also removed.
- The remaining individuals are given a fitness score by means of the following equation: $fitness = |wingap - W| + |AVGttf - A| + |AVGut - U| + (1 - MCTSwin) \cdot R$,

where $wingap$ is the difference between the number of wins of the best player and the number of wins of the worst player, $AVGttf$ is the average number of turns to finish the game, and $AVGut$ is the average number of turns where no action was taken because no player could satisfy the antecedent of any rule (useless turns). W is the target balance score, the desired value for $wingap$. A is the target number of turns to finish, and U is the target number of useless turns. Finally, every game is run again during M playouts. This time, the first player is simulated by a "vanilla" Monte Carlo Tree Search [2] agent without any particular optimisations. The other two remain as random agents. Thereafter, $MCTSwin$

is the rate of wins of the first player against the other two. This important feature measures the skill differentiation of the evaluated game, the probability of a player with superior skill (the MCTS agent) to beat a player of inferior skill (the random agent). R is a pre-specified scaling factor to weight the effect of this last feature within the fitness function. The goal of the evolutionary system is to minimize the fitness function, therefore the best fitness score is 0. If the size of the population is lower than I after a generation, it is refilled by adding randomly generated individuals after replacement.

4. RESULTS

We ran several evolutionary runs with $P = 3, T = 2, I = 100, E = 100, M = 20, W = 25, A = 30, U = 9, R = 25$ and target fitness set to 20. Table 1 shows the rules of a game called "The ant and the grasshopper" which is one of the more interesting games generated by the system. While several games show strong influences from one or two of the seed games (e.g. we found a twisted version of Blackjack with interesting gameplay), this is one of the more complete games not resembling any of the seed games. It is played as follows: Players always get the same amount of tokens due to the "GIVE" rules spread through several stages of the game. Although there are several chances of betting tokens during the gameplay (rule 1 in stage 4, and rules 0 and 3 in stage 5), there is no rule where players can gain tokens bet by other players. This means that the highest amount of tokens a player can get is the one provided by the game. Besides, there are no rules that kick players out of the game, so all of them make it to the end. Players get 5 victory points per token they have, consequently the one with the most tokens (bet tokens do not count) wins. Therefore, skipping "BET" rules and keeping as many tokens as possible is the best strategy to win. This game scored 13 points in fitness evaluation. Although the average number of turns to finish matched the target value (A), the average number of useless turns differed 12 points from the desired value U. This is due to rules involving table location 1 in their antecedent. Conditions related to that location are never fulfilled because the location remains empty during the whole game. Therefore, their related consequences can not be applied which may lead players to experiment useless turns. The win rate of the MCTS agent was 0.25, which shows that even a relatively weak player can beat random agents despite the apparent randomness of the game.

Table 1: Codification of "The Ant and the Grasshopper."

THE ANT AND THE GRASSHOPPER	
Stages and rules	
Stage 0	
COMPUTER COMMAND <Unconditional> GIVE Player: 0 Amount: 89 tokens	
COMPUTER COMMAND <Unconditional> DEAL Table: 0 Amount: 1 cards	
COMPUTER COMMAND <Unconditional> GIVE Player: 2 Amount: 39 tokens	
COMPUTER COMMAND <Unconditional> GIVE Player: <all> Amount: 87 tokens	
Stage 1	
if SHOW >= T0 then PLAY_IT	
COMPUTER COMMAND <Unconditional> DEAL Player: <all> Amount: 6 cards	
COMPUTER COMMAND <Unconditional> GIVE Player: <all> Amount: 58 tokens	
PLAY ONLY ONCE if SHOW SAME_RANK T1 then PLAY_IT	
Stage 2	
if SHOW < T1 then PLAY_IT	
PLAY ONLY ONCE if SHOW >= T0 then PLAY_IT	
PLAY ONLY ONCE if SHOW > T0 then PLAY_IT	
Stage 3	
COMPUTER COMMAND <Unconditional> GIVE Player: 0 Amount: 77 tokens	
Stage 4	
COMPUTER COMMAND <Unconditional> GIVE Player: 0 Amount: 44 tokens	
MANDATORY if PLAY 994, > T0 then BET	
Stage 5	
PLAY ONLY ONCE if DRAW then BET	
if SHOW >= T0 then PLAY_IT	
COMPUTER COMMAND <Unconditional> GIVE Player: <all> Amount: 63 tokens	
PLAY ONLY ONCE if DRAW then BET	
Ranking	
Card(s)	Value
Four of a kind	190
6 + 8 + Jack	212
Winning conditions	
5 points for each token. 3 points for finishing the game.	

5. DISCUSSION

Although for brevity only one game could be shown, we have generated dozens and analysed a handful in detail. It seems that almost every run generates one or more games that are well-formed, always lead to one player winning and the others losing if played for long enough, are balanced when played by random agents, and allow skill differentiation. The game descriptions are human-readable, so with appropriate equipment (a deck and a few Euro) they can be played by humans. We therefore assert that the goals for this particular paper, as set out in section 1, have been fulfilled.

On the other hand, these games are not entertaining for a human to play, and they do not include any particularly interesting novel mechanics. Every rule set comprises several strategies that can be followed by players, including one that leads to the victory. Nevertheless, these strategies turn out to be neither very exciting nor challenging. In some cases, a game can be won with a single action in the first turn. In response to this, a fitness function could be designed that rewards games for requiring a minimum diversity of different actions taken in a sequence to be won. An alternative solu-

tion is to use a more elaborate skill differentiation measure which rewards not only that weak players win over random players, but also that strong players win over weaker players.

The generated games still occasionally contain rules including unfulfillable antecedents, or locations that are filled with cards turns after they are referenced within a condition. In response to this, additional fitness functions could be devised that penalise rulesets which include rules that are never used. Note that in the cascading framework, it is generally easy to "stack" fitness functions. At the top of a future stack of such functions we would like to test for rules that endorse engagement or "fun". This will almost certainly require the development of several different game playing agents with differing skill and playing style, and possibly learning abilities. There are several ideas developed for judging the quality of other games which could be adapted to card games, including learnability [15] and various measures of tension and lead changes [1].

6. CONCLUSION

This paper presented an evolutionary system that automatically generates card games. A grammar $G_{cardgame}$ was used to generate a language whose strings codify well-formed card games, including codifications of Texas hold 'em poker, blackjack and UNO. This grammar is used by a grammar-guided genetic program, that applies selection, crossover and mutation to a combined random/fixed initial population of individuals. A cascading fitness function was designed in order to address different goals, including checking that games are playable, non-drawish, not too ruled, balanced and skill-differentiated. To our knowledge, this is the first time anyone has automatically generated card games.

7. REFERENCES

[1] C. Browne and F. Maire. Evolutionary game design. *Computational Intelligence and AI in Games, IEEE Transactions on*, 2(1):1–16, 2010.

[2] C.B. Browne, E. Powley, D. Whitehouse, S.M. Lucas, P.I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A survey of monte carlo tree search methods. *Computational Intelligence and AI in Games, IEEE Transactions on*, 4(1):1–43, 2012.

[3] Michael Cook and Simon Colton. Multi-faceted evolution of simple arcade games. In *Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG)*, 2011.

[4] J. M. Font. Evolving third-person shooter enemies to optimize player satisfaction in real-time. In Cecilia et al. Di Chio, editor, *Applications of Evolutionary Computation*, volume 7248 of *Lecture Notes in Computer Science*, pages 204–213. Springer Berlin / Heidelberg, 2012.

[5] J. M. Font, T. Mahlmann, D. Manrique, and J. Togelius. A card game description language. Accepted for publication in Evostar 2013, April 2013.

[6] J. M. Font and D. Manrique. Grammar-guided evolutionary automatic system for autonomously building biological oscillators. In *2010 IEEE Congress on Evolutionary Computation*, pages 1–7, July 2010.

[7] J. M. Font, D. Manrique, and J. Ríos. Evolutionary construction and adaptation of intelligent systems. *Expert Systems with Applications*, 37:7711–7720, 2010.

[8] M. Garcia-Arnau, D. Manrique, J. Rios, and A. Rodriguez-Paton. Initialization method for grammar-guided genetic programming. *Knowledge-Based Systems*, 20(2):127–133, 2007.

[9] Vincent Hom and Joe Marks. Automatic design of balanced board games. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, pages 25–30, 2007.

[10] Nathaniel Love, Timothy Hinrichs, David Haley, Eric Schkufza, and Michael Genesereth. General game playing: Game description language specification. Report LG-2006-01, Stanford Logic Group, Computer Science Department, Stanford University, Stanford, CA, March 4 2006.

[11] Tobias Mahlmann, Julian Togelius, and Georgios Yannakakis. Modelling and evaluation of complex scenarios with the strategy game description language. In *Proceedings of the Conference on Computational Intelligence and Games (CIG) 2011*, Seoul, KR, 2011.

[12] Mark J. Nelson. Encoding and generating videogame mechanics. Tutorial at the 2012 IEEE Conference on Computational Intelligence and Games. http://www.kmjn.org/notes/generating_mechanics_bibliography.html, 2012.

[13] Adam M. Smith and Michael Mateas. Variations forever: Flexibly generating rulesets from a sculptable design space of mini-games. In *Proceedings of the IEEE Conference on Computational Intelligence and Games*, pages 273–280, Copenhagen, Denmark, 18–21 August 2010.

[14] J. Togelius, R. De Nardi, and S.M. Lucas. Towards automatic personalised content creation for racing games. In *Computational Intelligence and Games, 2007. CIG 2007. IEEE Symposium on*, pages 252–259. IEEE, 2007.

[15] J. Togelius and J. Schmidhuber. An experiment in automatic game design. In *Computational Intelligence and Games, 2008. CIG'08. IEEE Symposium On*, pages 111–118. IEEE, 2008.